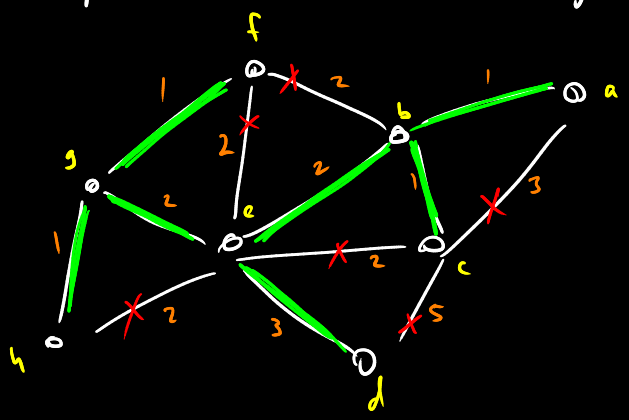
Last Time: Prim's Algorithm and Kruskal's Algorithm.

↳ Given a weighted graph (G, w)

↳ Compute a minimum weight spanning tree in G.

Ex: Compute a minimum weighted spanning tree in



i	e_i	w_i
1	gf	1
2	bc	1
3	ab	1
4	gh	1
5	ge	2
6	be	2
7	de	3

The highlighted tree is a minimum weight spanning tree in (G, w) (constructed via Kruskal's algorithm with the table above).

This minimum-weight spanning tree has weight 11.

Prüfer Code

Q: How many trees are there?

↳ Given a vertex set V $(= \{0\} \cup [n])$, how many graphs on V are trees?

↳ Consider K_{n+1}. How many spanning trees does K_{n+1} have?

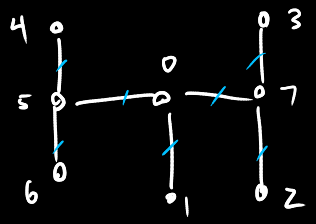
A: Point of this lecture...

We will count the number of labelled trees on $n+1$ vertices (i.e. w/ n edges) using a bijection:

$$\{\text{labelled trees}\} \longleftrightarrow \{\text{sequences in } \{0\} \cup [n]\}.$$

↑ Goal...

Ex:



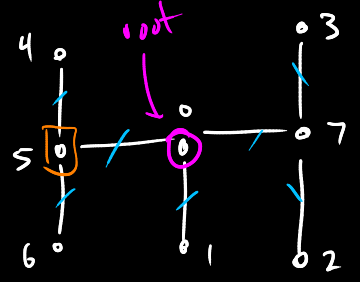
one option: matrix of edges:

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 3 & 4 & 5 \\ 1 & 5 & 7 & 7 & 7 & 5 & 6 \end{pmatrix}$$

↑ columns of this array describe edges.

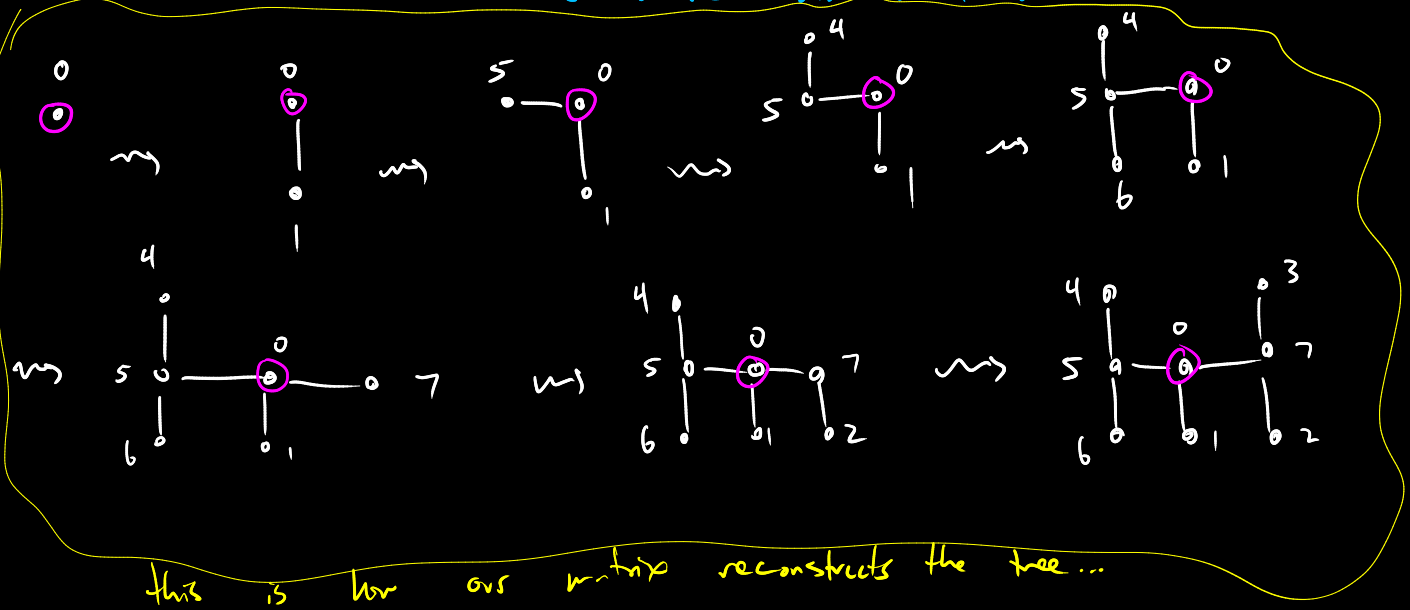
Note: doesn't accomplish the goal...

→ Hard to classify/understand which matrices are "allowed" here...



$$\begin{pmatrix} 0 & 0 & 5 & 5 & 0 & 7 & 7 \\ 1 & 5 & 4 & 6 & 7 & 2 & 3 \end{pmatrix}$$

↑ encodes the same tree as before, but has more data about what branches look like...

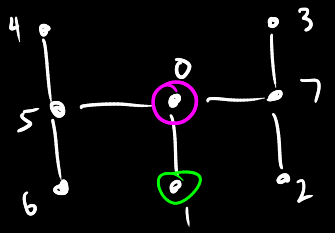


this is how our matrix reconstructs the tree...

Still inefficient, but it's better than before...

IDEA: Deconstruct the tree from the top down...

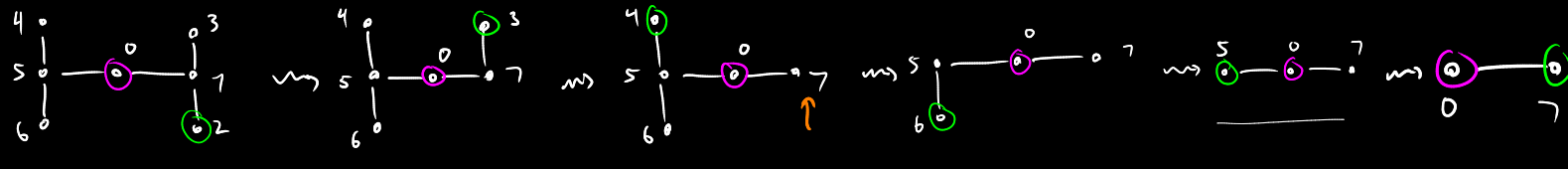
↪ Remove leaves...



$$\begin{matrix} \text{leaf} \to \\ \text{parent} \to \end{matrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 6 & 5 & 7 \\ 0 & 7 & 7 & 5 & 5 & 0 & 0 \end{pmatrix}$$

↑ Encodes the tree via "smallest leaf at stage i"...

↑ chop off and record edge



Q: What do we know about the resulting array?

① The top row (i.e. row of leaves) has no repeats (because we remove each leaf as we see it...)

② the top row doesn't include 0 (b/c 0 is our root, so it is Never a leaf...)

③ The top row (of leaves) is a permutation of $[n]$ b/c ① and ②.

④ The last column always has the form $\binom{K}{0}$ (b/c 0 is never removed, and we are recording the final leaf edge).

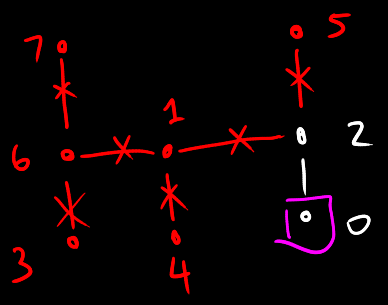
⑤ The top row of the array can be reconstructed from the bottom row (IDEA: "smallest available entries").

Illustration of 5: $(0,7,7,5,5,0)$ ← Prüfer code of the tree

$$\rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 6 & 5 & 7 \\ 0 & 7 & 7 & 5 & 5 & 0 & 0 \end{pmatrix}$$

↑ "cut off column"

Ex:



✱ At stage i: identify smallest leaf, remove it, and record it's parent.

$(6,1,2,6,1,2)$

NB: 0 is always the root vertex for our construction.

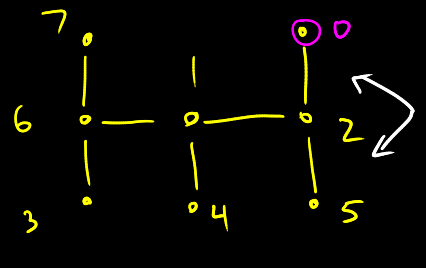
Reconstruction:

$(6,1,2,6,1,2)$ → Start by adding a 0 at the end.

→ at step i: add back smallest possible label to top row.

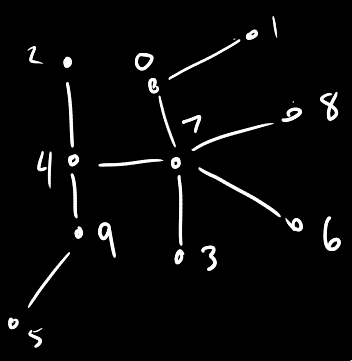
↳ $\begin{pmatrix} 3 & 4 & 5 & 7 & 6 & 1 & 2 \\ 6 & 1 & 2 & 6 & 1 & 2 & 0 \end{pmatrix}$ ← matrix of leaf-edges:

Picture:



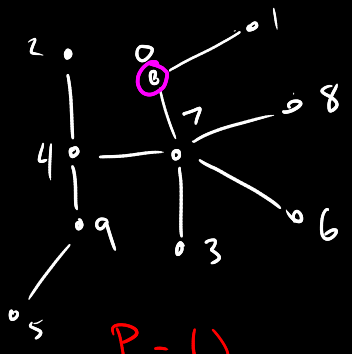
← Same as before ☺

Ex: Compute the Prüfer code of the labelled tree below:

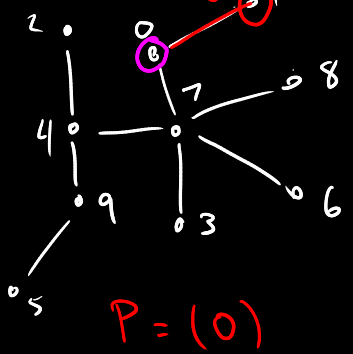


Sol: We deconstruct the tree as follows (tracking the Prüfer code as we go).

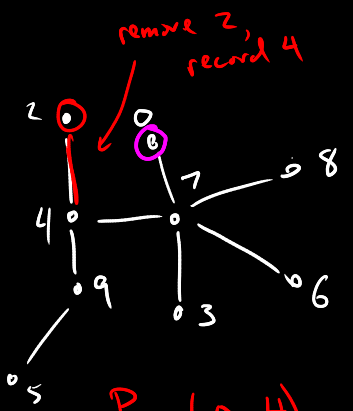
(Remember: 0 is root, remove smallest leaves, record their neighbours, stop at last leaf.)



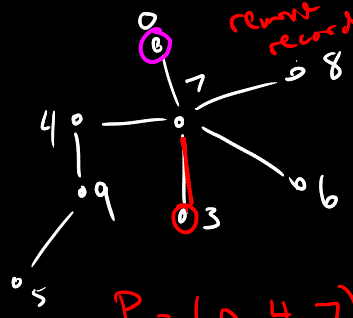
$P = ()$



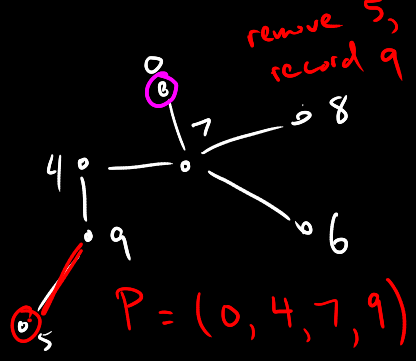
$P = (0)$



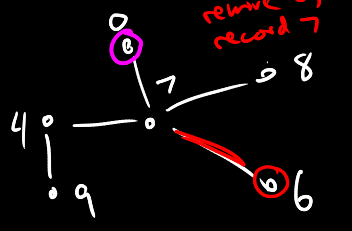
$P = (0, 4)$



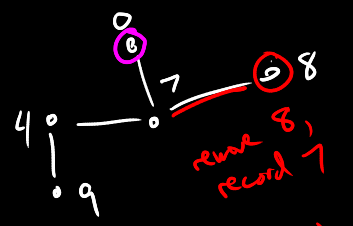
$P = (0, 4, 7)$



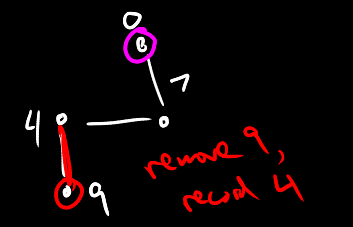
$P = (0, 4, 7, 9)$



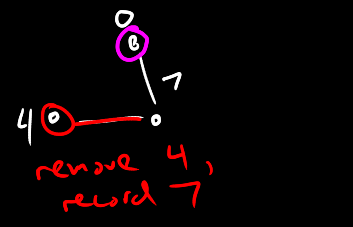
$P = (0, 4, 7, 9, 7)$



$P = (0, 4, 7, 9, 7, 7)$



$P = (0, 4, 7, 9, 7, 7, 4)$



$P = (0, 4, 7, 9, 7, 7, 4, 7)$

Hence the Prüfer code of this labeled tree is $(0,4,7,9,7,7,4,7)$.

Exercise: Check this Prüfer code is correct using the reverse construction

Ex: What labelled tree is represented by the Prüfer code $(0,1,1,3,1,5,7,2)$?

Sol: We construct the edge-matrix of T first:

(Remember: add a 0, make this the bottom row of the matrix and record the smallest possible leaf-nodes.)

$$\begin{pmatrix} * & * & * & * & * & * & * & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & * & * & * & * & * & * & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & * & * & * & * & * & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & 8 & * & * & * & * & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & 8 & 9 & * & * & * & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & 8 & 9 & 3 & * & * & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & 8 & 9 & 3 & 1 & * & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & 8 & 9 & 3 & 1 & 5 & * & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & 8 & 9 & 3 & 1 & 5 & 7 & * \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 6 & 8 & 9 & 3 & 1 & 5 & 7 & 2 \\ 0 & 1 & 1 & 3 & 1 & 5 & 7 & 2 & 0 \end{pmatrix}$$

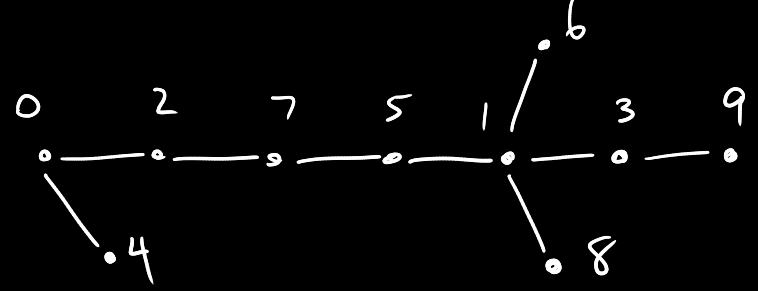
= unavailable

$*$ = unknown

= new

= ignored

$\therefore$ the desired tree is



Exercise: Compute the Prüfer code of this tree to check...

The Prüfer code provides us the answer to our question:

Prop (Cayley's Theorem for Trees): There are $(n+1)^{n-1}$ labelled trees with $n+1$ vertices.

Sketch: Each sequence in $\{0\} \cup [n]$ of length $n-1$ is the Prüfer code of a unique labelled tree. Because each labelled tree also has a Prüfer code, we have a bijection

$$\{T : T \text{ is a tree with } V(T) = \{0\} \cup [n]\} \longleftrightarrow (\{0\} \cup [n])^{n-1}.$$

Hence the result follows. ▨

Thanks for your attention this semester :)

– Chris E